17th International Conference on Reliable Software Technologies

ADA-EUROPE 2012

11-15 June 2012, Stockholm, Sweden

Combining Ada Generics and Code Generation to implement a Software Product Line

Ada Europe Industrial Talk
Stockholm
June 14, 2012

EUROCOPTER
AN EADS COMPANY

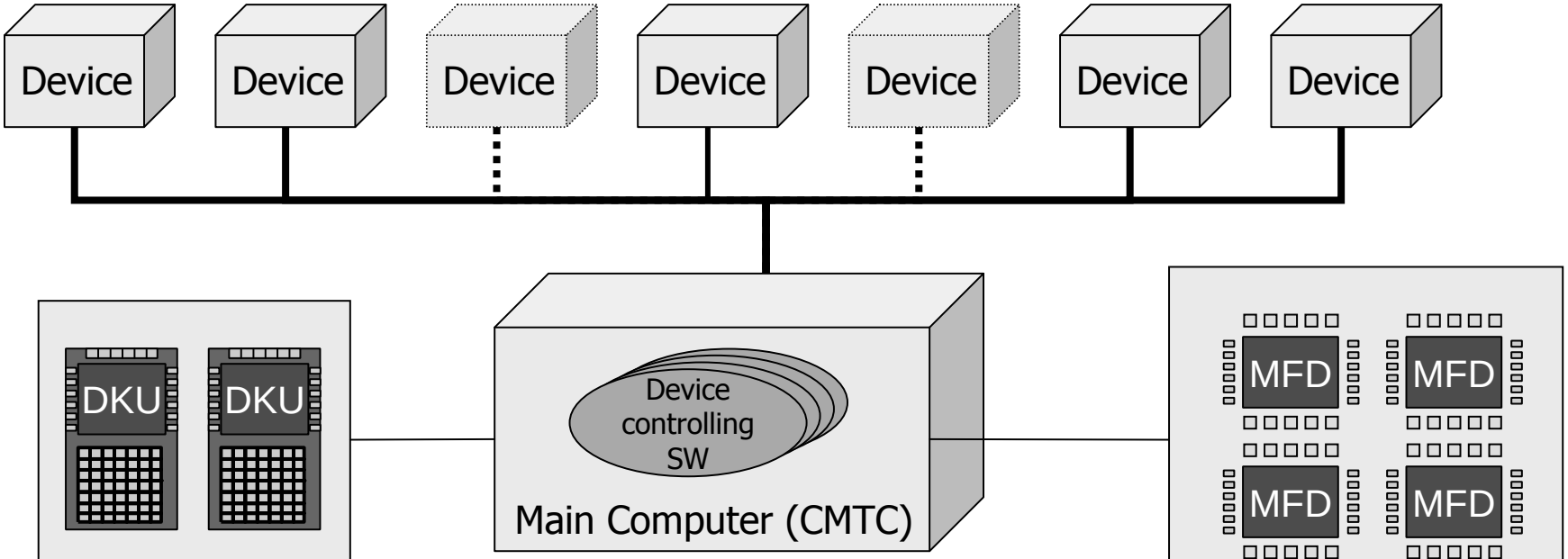
ESG

Project Background NH90

- Medium weight multi-role military helicopter
- Two basic versions:
  - Tactical Transport Helicopter (TTH)
  - NATO Frigate Helicopter (NFH).
- More than 500 NH90s ordered in 23 variants

Avionics System Architectures

Product variants defined by differing sets of equipment/devices (system configurations):



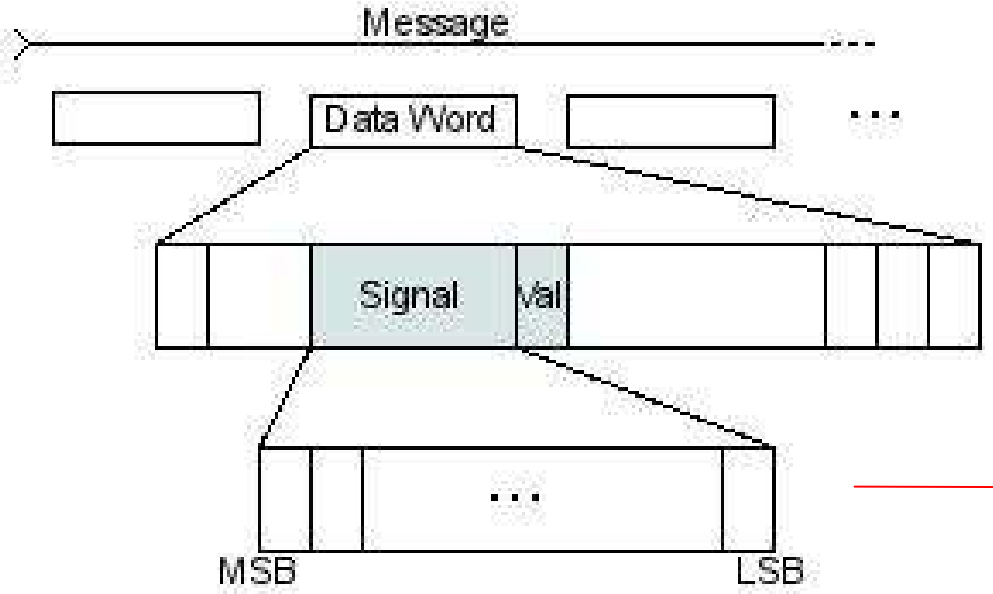
- Varying number of DKUs
- Varying DKU format definitions depending on system configuration

- Varying sets of device controlling SW
- Customizations to System Monitoring
- Varying MILBUS and other message handling protocols
- ...

- Varying number of MFDs
- Varying MFD format definitions depending on system configuration

Note: NH90 project specific safety level 2 – between DAL B and C (ARP 4754).

Messages and Signals



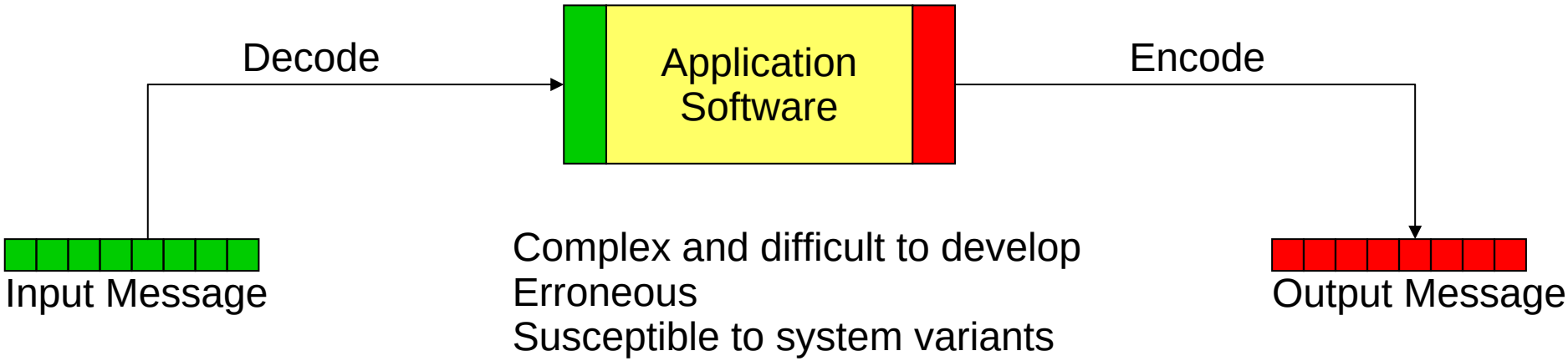
MIL-1553: up to 32 20 bit data words with 16 bit payload

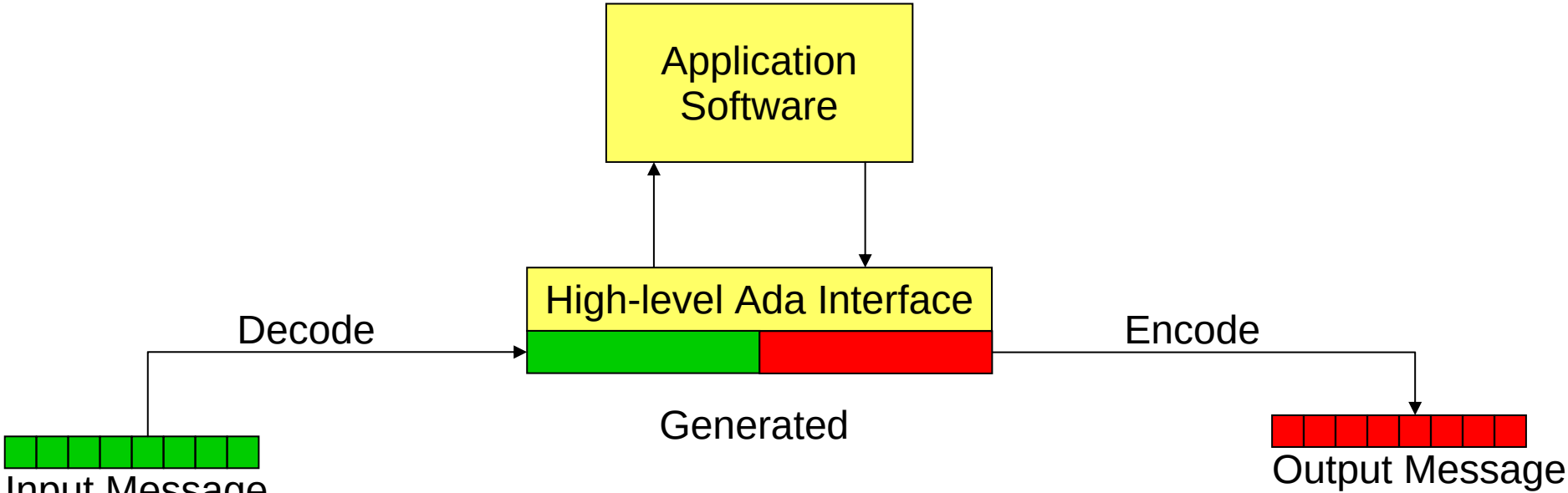
ARINC 429: one 32 bit word with 19 bit payload

Signals are typed

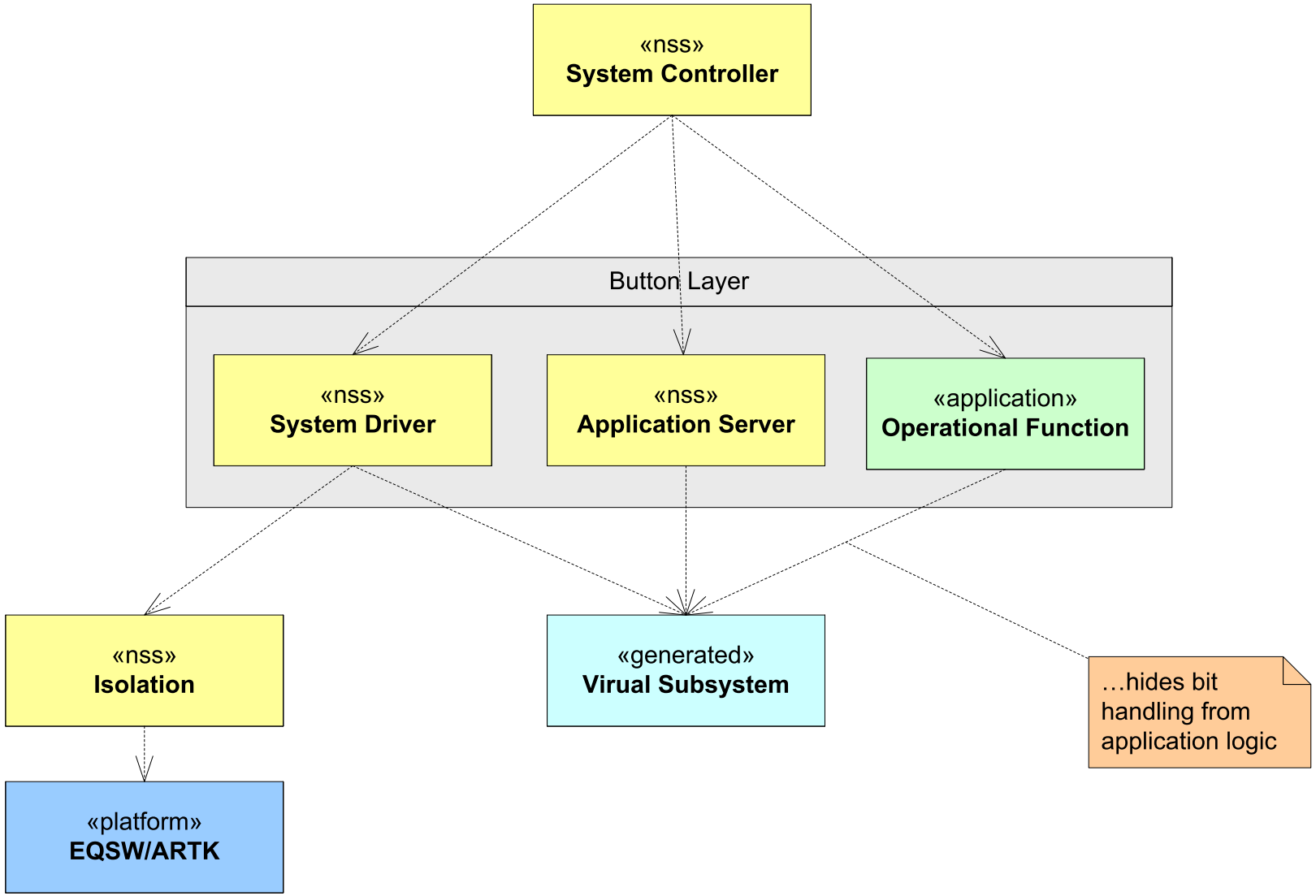
Avionic Bus Signals	CMC		MTC	
	IN	***OUT***	***IN***	***OUT***
MIL-1553	13500	26600	5500	7000
ARINC 429	500	1200	300	2700

Message Data Processing

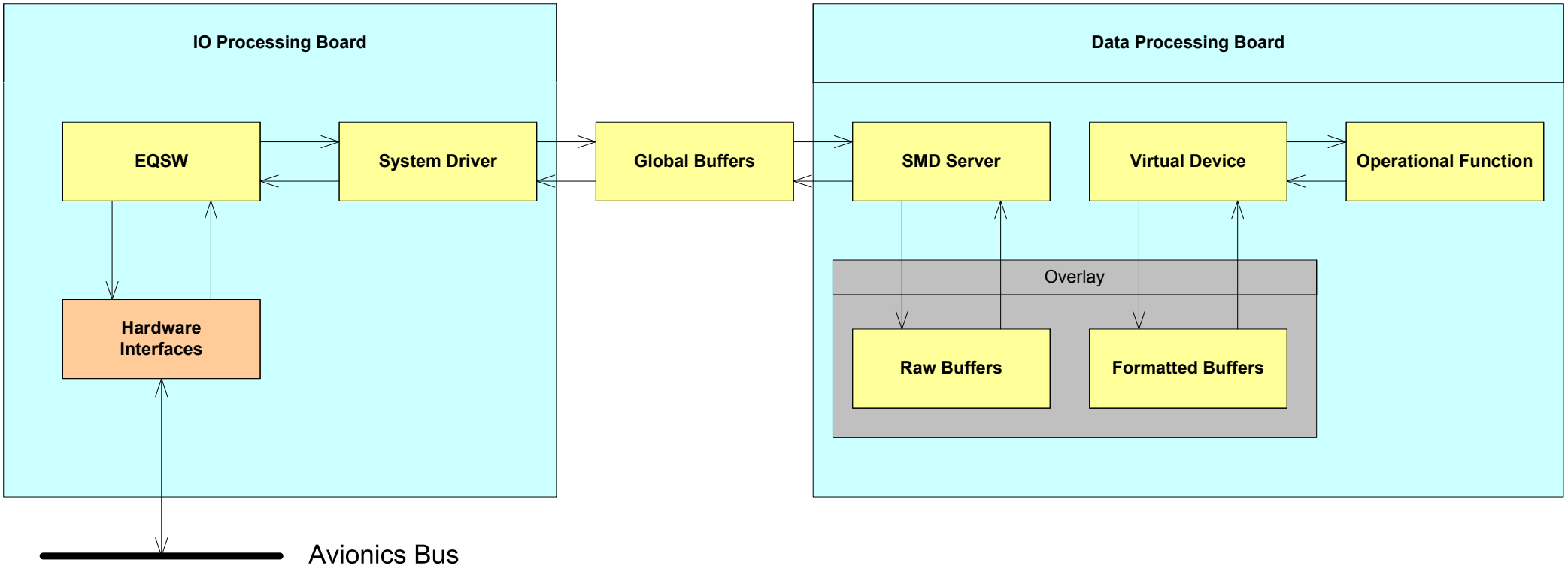




Software Architecture



IO Processing



Virtual Device – Operational IF

```
with ...
package Flir_Vd is

    procedure Get_Flir_Los_Data (Value : out Flir_Types.Flir_Los_Data;
                                 Valid : out General_Types.Data_Validity);

    procedure Put_Flir_Mode_Commands (Value : in
                                      Flir_Types.Flir_Mode_Commands);

    ...
end Flir_Vd;
```

Equipment or Device represented by VD

SW object for FLIR Device

High Level SW Type
One of: scalar, record, array, array of record

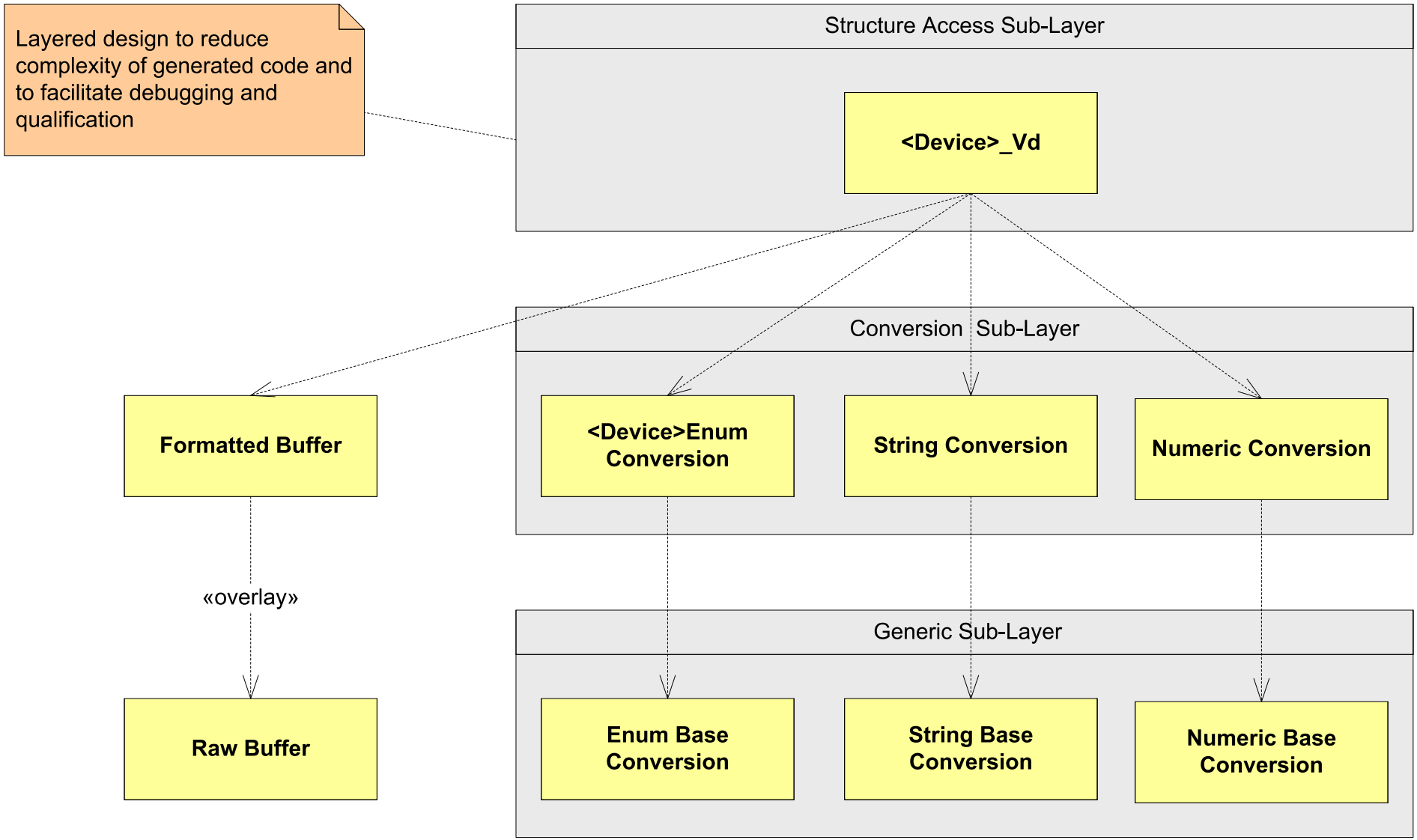
Explicit vaildity information

SW Object representing Commands sent to Device

Enumeration Types contain a Not_Valid literal by convention.

A *virtual device* (VD) acts as high level facade to data received from and sent to a device.

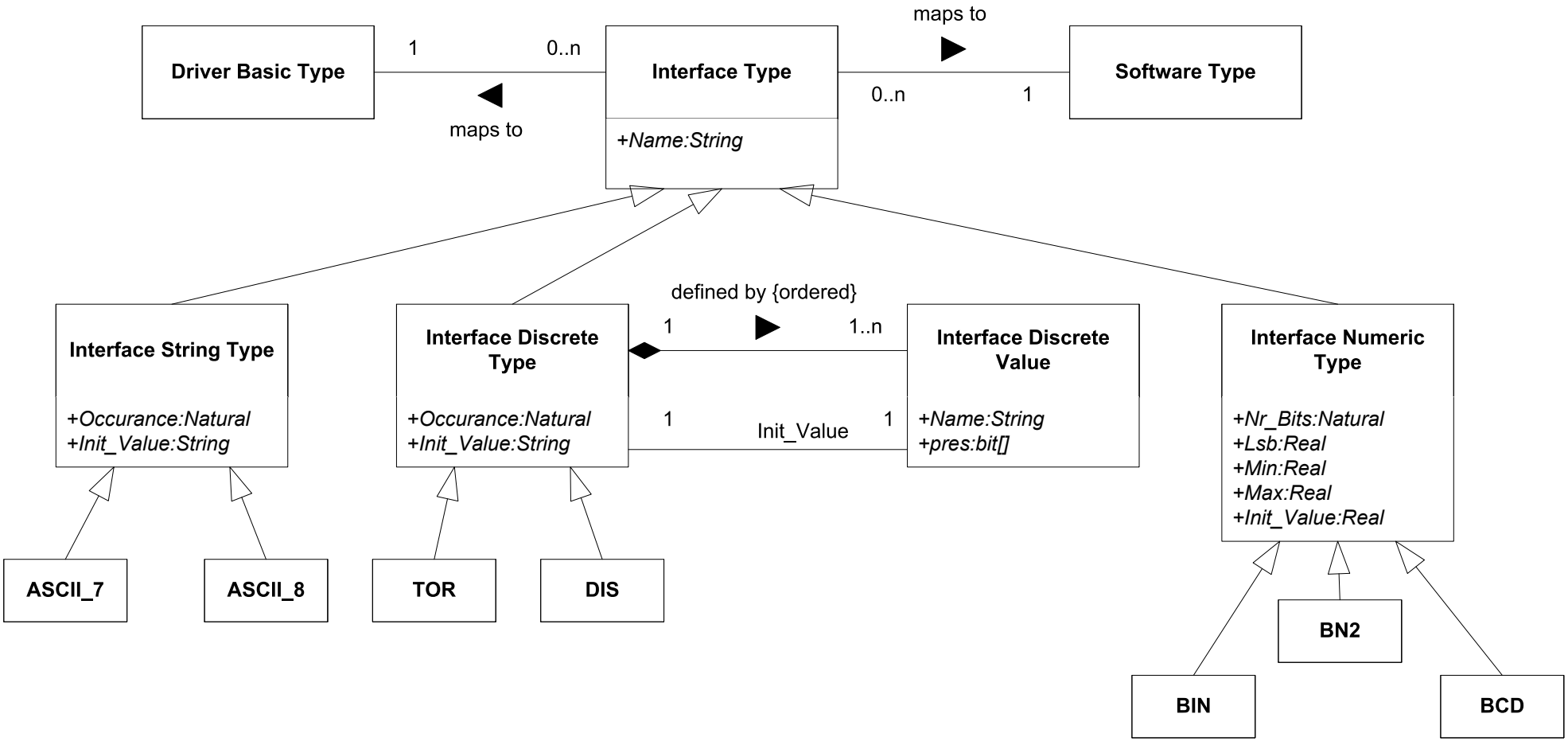
Virtual Subsystem Components



Buffer Overlay

```
Mission_Flir_Los_Data_Formatted : Milbus_Formatted_Types.Flir_Los_Data;
 for Mission_Flir_Los_Data_Formatted use at Mission_Flir_Los_Data_Local'Address;


 type Flir_Los_Data is
 record
     ...
     Flir_Mtcc_Los_Data_Valid : T_B_Validity_00;
     ...
     Flir_Mtcc_Los_Dat_Az : Raw_Int_16;
     Flir_Mtcc_Los_Dat_El : Raw_Int_16;
 end record;

 for Flir_Los_Data use
 record
    ...
    Flir_Mtcc_Los_Data_Valid at 0 range W16_Bit_2_2'First .. W16_Bit_2_2'Last;
    ...
    Flir_Mtcc_Los_Dat_Az at 2 range W16_Bit_0_15'First .. W16_Bit_0_15'Last;
    Flir_Mtcc_Los_Dat_El at 4  range W16_Bit_0_15'First .. W16_Bit_0_15'Last;
 end record;

 for Flir_Los_Data'Size use 48;
```

Signal Data Types



Generics Sub layer

Class	Generic Conversion Package (Base Conversions)	Interface Type	Software Type
Numeric	Bin_Int_Generic	BIN/BN2	Integer Type
	Bin_Real_Generic	BIN/BN2	Real (Float) Type
	Bcd_Int_Generic	BCD	Integer Type
	Bcd_Real_Generic	BCD	Real (Float) Type
Discrete	Bool_Generic	TOR	Boolean
	Enum_Generic	DIS	Enumeration
String	Str_Ascii7_Generic	ASCII7	String Type
	Str_Ascii8_Generic	ASCII8	String Type

Base Conversions

```
generic
     type Raw is range <>;
     Raw_Min : Raw;
     Raw_Max : Raw;
     Raw_Default : Raw;
     --
     type Eng is digits <>;
     Eng_Min: Eng;
     Eng_Max: Eng;
     Eng_Default : Eng;
     --
     Lsb : Universal_Types.Real;

package Bin_Real_Generic is

     type Image_With_Val is
         record
          Value : Eng;
          Valid : Boolean;
         end record;

     type Raw_With_Val is
         record
          Value : Raw;
          Valid : Boolean;
         end record;

     function To_Image (Source : in Raw;  Validity : in Boolean) return Image_With_Val;

     function To_Raw (Source : in Eng;  Validity : in Boolean ) return Raw_With_Val;

end Bin_Real_Generic;
```

Imported driver type

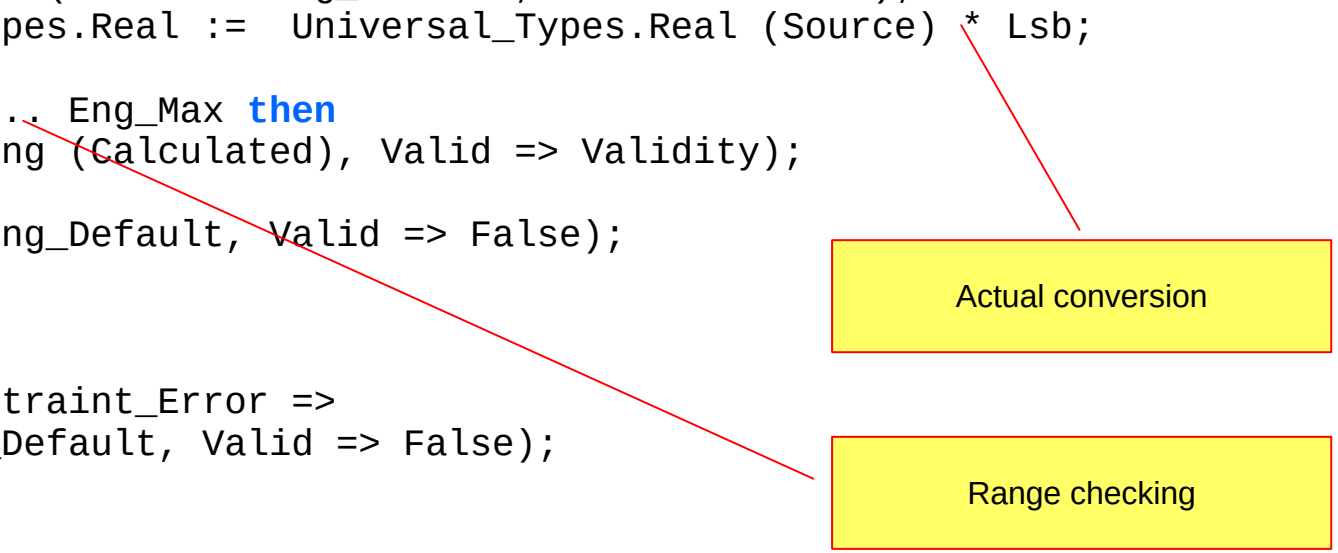
Imported SW type

Record types used as return values for functions

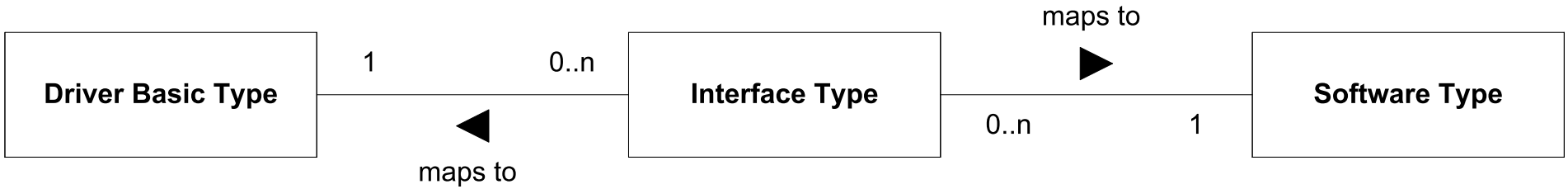
Conversion Functions

Generic Conversion Functions

```
function To_Image (Source : in Raw;  Validity : in Boolean ) return Image_With_Val is
    Result : Image_With_Val := (Value => Eng_Default, Valid => False);
    Calculated : Universal_Types.Real :=  Universal_Types.Real (Source) * Lsb;
begin
    if Calculated in Eng_Min .. Eng_Max then
        Result := (Value => Eng (Calculated), Valid => Validity);
    else
        Result := (Value => Eng_Default, Valid => False);
    end if;
    return Result;
exception
    when Numeric_Error | Constraint_Error =>
        return (Value => Eng_Default, Valid => False);
end To_Image;
```



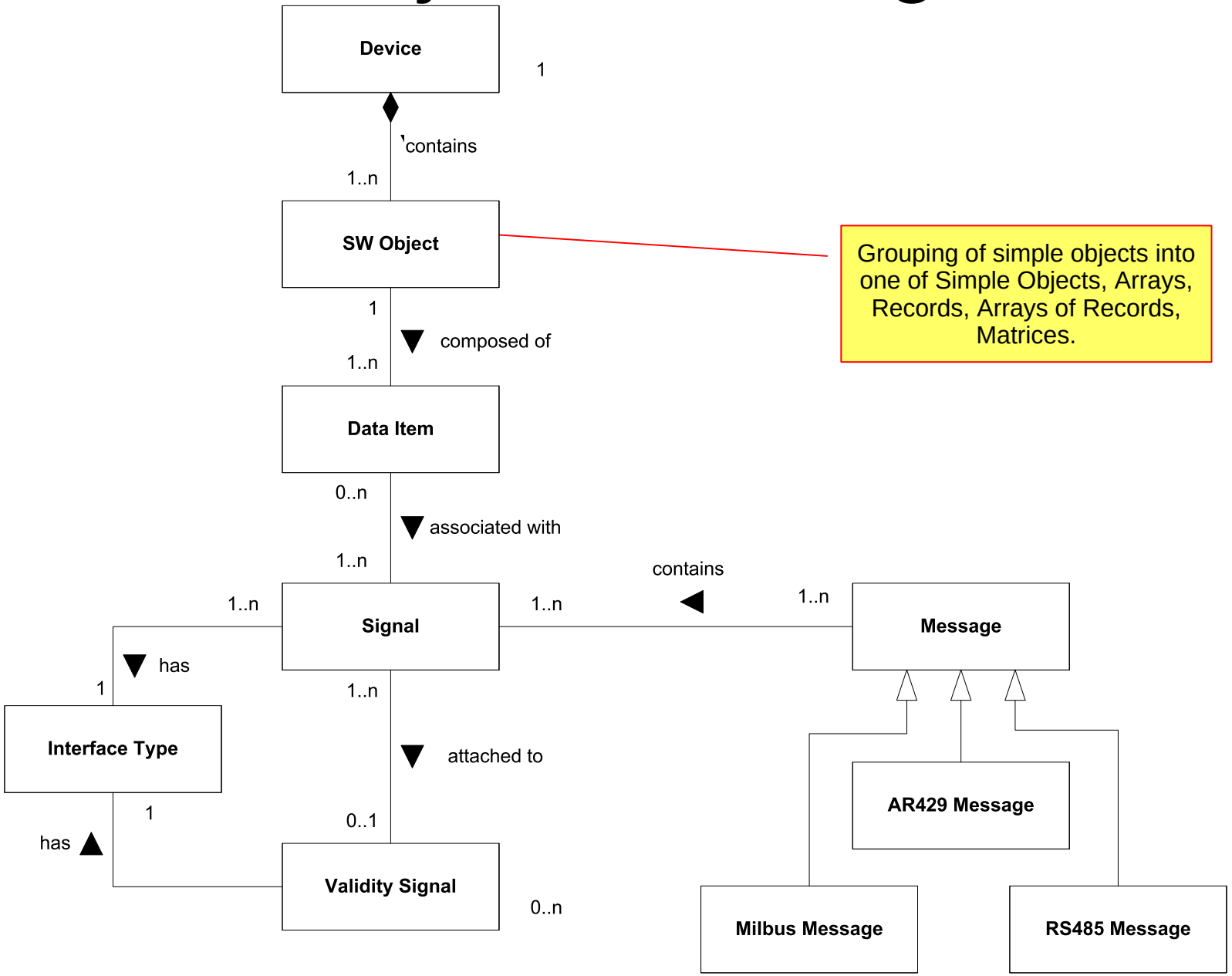
Conversion Sublayer



There is an instantiation of a base conversion package for every used interface type and its associated SW type.

```
package T_I_Mission_Angle is
   new Drivers_Base_Conversions.Bin_Real_Generic
      (Raw => Drivers_Basic_Types.Raw_Int_16,
       Raw_Min => Drivers_Basic_Types.Raw_Int_16 (-32768),
       Raw_Max => Drivers_Basic_Types.Raw_Int_16 (32767),
       Raw_Default => Drivers_Basic_Types.Raw_Int_16 (0),
       Eng => Universe.Degree,
       Eng_Default => Universe.Degree (0.0),
       Lsb => Universal_Types.Real (0.00549317));
```

SW Objects and Signals



Structured Access Layer

```
package body Flir_Vd is
. . .
    procedure Get_Flir_Los_Data (Value : out Flir_Types.Flir_Los_Data;
                  Valid : out General_Types.Data_Validity) is
        Temp_Los_Azimuth : T_I_Mission_Angle.Image_With_Val :=
            T_I_Mission_Angle.To_Image
                (Mission_Flir_Los_Data_Formatted.Flir_Mtcc_Los_Dat_Az,
                 Mission_Flir_Los_Data_Validity and
                To_Bool
                    (Mission_Flir_Los_Data_Formatted.Flir_Mtcc_Los_Data_Valid);
        Temp_Los_Elevation : T_I_Mission_Angle.Image_With_Val :=
            T_I_Mission_Angle.To_Image
                (Mission_Flir_Los_Data_Formatted.Flir_Mtcc_Los_Dat_El,
                 Mission_Flir_Los_Data_Validity and
                To_Bool
                    (Mission_Flir_Los_Data_Formatted.Flir_Mtcc_Los_Data_Valid);
    begin
        Value := (Los_Azimuth => Temp_Los_Azimuth.Value,
                  Los_Elevation => Temp_Los_Elevation.Value);
        Valid := Temp_Los_Azimuth.Valid and Temp_Los_Elevation.Valid;
    end Get_Flir_Los_Data;
. . .
end Flir_Vd;
```

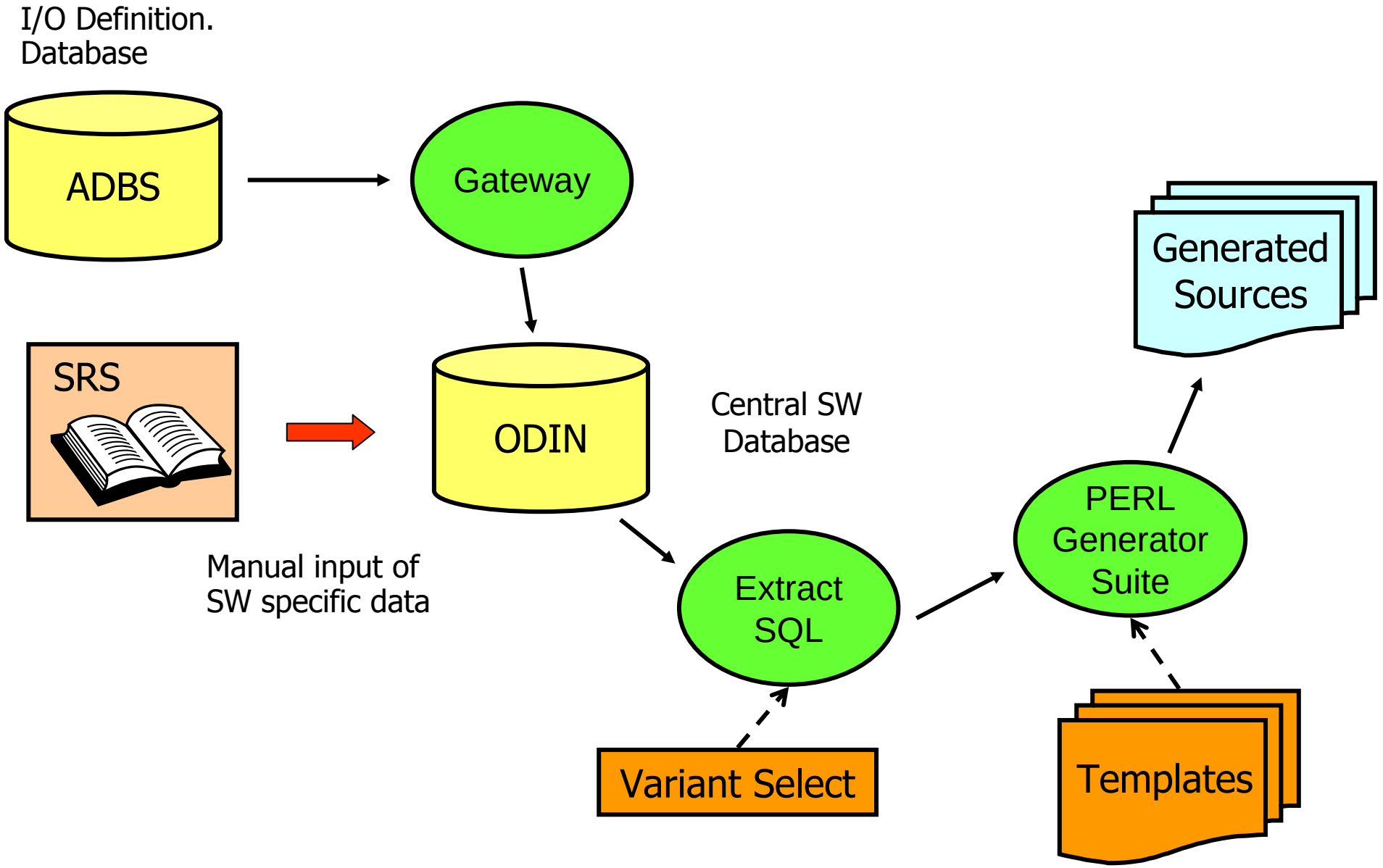
Coding standard requires all local variables to be initialized – so conversion is performed in declaration part.

Conversion function from conversion sublayer.

Access to proper signal in formatted buffer.

Overall validity is a conjunction of signal and message validity.

Tool Chain Overview



Template Based Code Generation

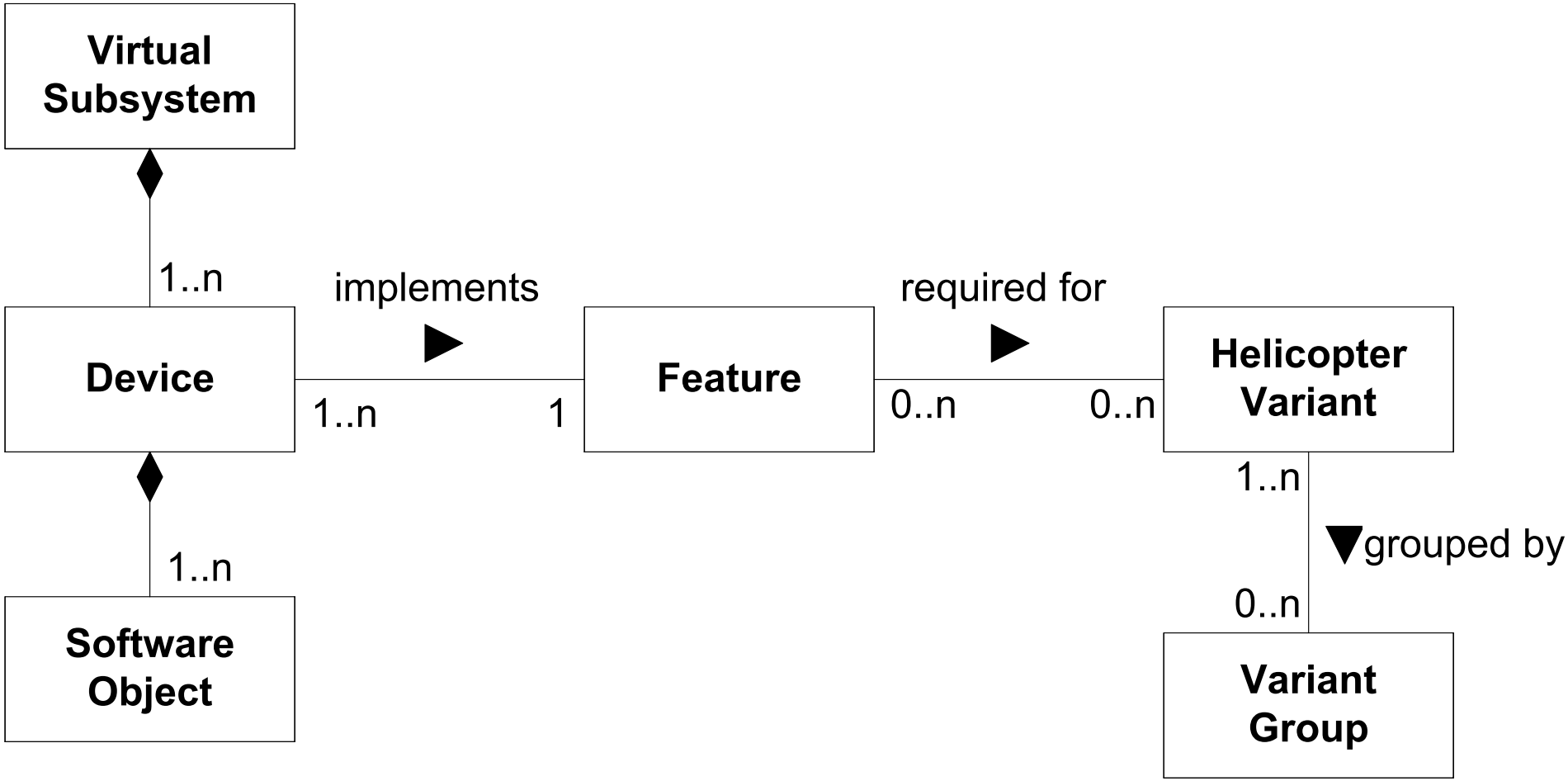
```
<tmpl_include name="Common/spec_header.tmpl">
-- @Purpose:
--    Package <tmpl_var name="device">_Vd provides the procedural
--    interface for <tmpl_var name="device"> Vd.
-------------------------------------------------------

<tmpl_loop name="withs"><NOBR>
with <tmpl_var name="name">;
</tmpl_loop><NOBR>

package <tmpl_var name="device">_Vd is
...

<tmpl_loop name="sw_object">

    -- @Purpose:
    --    This represents the procedural interface
    --    to the <tmpl_var name="suffix"> VD.
    --    ...

    <tmpl_if name="is_get"><NOBR>
        <tmpl_include name="SUBSYSTEM#DEVICE_lld.1.ada.get_function.tmpl">
    </tmpl_if><NOBR>
    <tmpl_if name="is_put"><NOBR>
        <tmpl_include name="SUBSYSTEM#DEVICE_lld.1.ada.put_function.tmpl">
    </tmpl_if><NOBR>
</tmpl_loop><NOBR>

end <tmpl_var name="device">_Vd;
```

Device Name

Loop over all SW Objects associated with Device

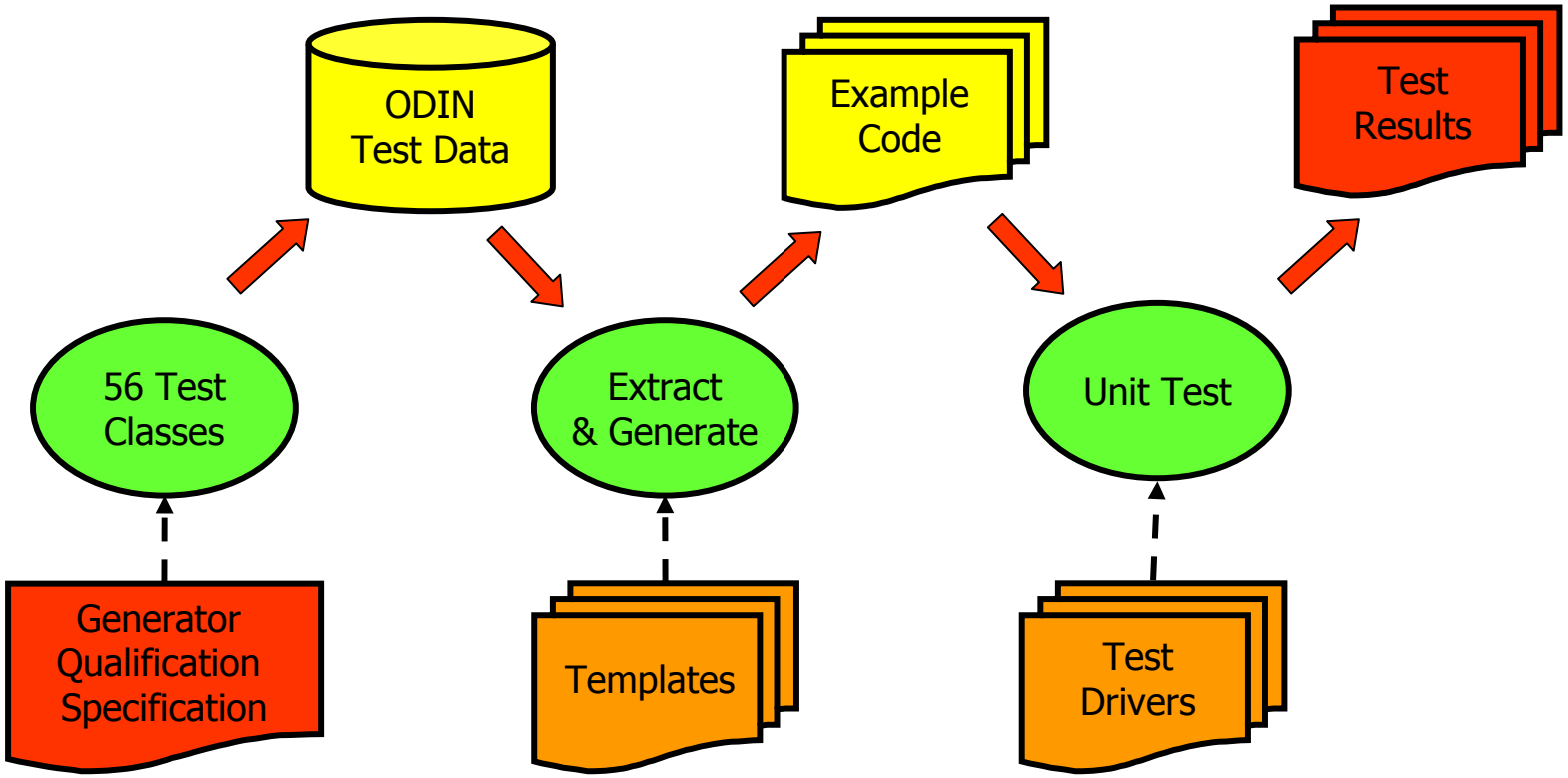
Put and Get Procedures for every Object in separate Templates.

Variants and Features



Code Generator Qualification

- Generator Tools must be qualified if they reduce, eliminate or automate processes.
- Generator Qualification Plan describes the qualification approach:



Summary

- High level access to low level interface data is a common problem in resource constrained embedded systems without OSI Stack
- Generation provides clean pure Ada representation of low level I/O data as a solution to that problem
- SW developers are relieved from error prone, time consuming low level bit manipulation
- Developers do not need to care about inclusion or exclusion of signals and messages for the different variants
- Majority of the code is generated
- Code Generators are qualified so there is no need to formally verify generated code.
- Standardized solution to a common problem across a many developers instead of individual solutions.

- Richard.Bridges@eurocopter.com
- Frank.Dordowsky@esg.de
- Holger.Tschoepe@eurocopter.com